PES University

Department of Computer Science & Engineering
Object Oriented Analysis and Design
Mini Project

UE22CS352B

Employee Management System

(OOAD Project)

Submitted by:

Vundela Vipul Kumar Reddy (PES1UG22CS709)
Veluru S L Dheeraj Chowdary (PES1UG22CS684)
Aditya KR (PES2UG22CS244)
Suhas PH (PES2UG22CS588)

April 22, 2025

1 Introduction

The **Employee Management System** is a modern web-based solution designed to digitize and streamline various Human Resource (HR) operations in an organization. This project aims to eliminate paperwork and manual effort by offering an integrated digital platform that manages employees across their lifecycle—from onboarding to exit.

The system encapsulates essential modules such as employee registration, payroll automation, leave tracking, and background verification. Each module is carefully designed to align with industry-standard practices, ensuring scalability and ease of use across small to large enterprises.

Built using a combination of React.js for the frontend and Spring Boot for the backend, the system embodies principles of modularity, responsiveness, and robustness. The architecture adheres to the **Model-View-Controller (MVC)** design paradigm and leverages multiple object-oriented principles and design patterns to ensure a maintainable and extensible codebase.

Technology Stack Overview:

- **Frontend:** Designed using **React.js**, the user interface is dynamic, component-based, and offers a smooth user experience. Forms, dashboards, and employee listings are updated in real-time using stateful components and REST API communication.
- **Backend:** Developed with **Spring Boot**, the backend ensures secure routing, validation, and business logic processing. It handles user authentication, employee CRUD operations, payroll processing, and verification workflows through modular RESTful services.
- **Database: MySQL** serves as the persistent storage layer. It stores employee data, leave applications, payroll history, uploaded documents, and verification statuses using normalized relational schemas.

In addition to core CRUD operations, the system supports features like:

- Automatic salary computation and payroll generation
- Leave request workflows with approval/rejection by admins
- Real-time status updates on document verification

By separating concerns between UI rendering (React), business logic (Spring Boot), and data persistence (MySQL), the architecture remains flexible for enhancements like role-based access control, notification services, and third-party integrations.

In summary, this project serves as a comprehensive case study for applying object-oriented design and system architecture principles in a real-world HR automation solution.

2 Project Design

To ensure the system meets both functional and non-functional requirements, a series of UML diagrams were developed during the analysis and design phase.

These diagrams visually communicate the system's behavior, structure, and user interactions:

2.1 Use Case Diagram

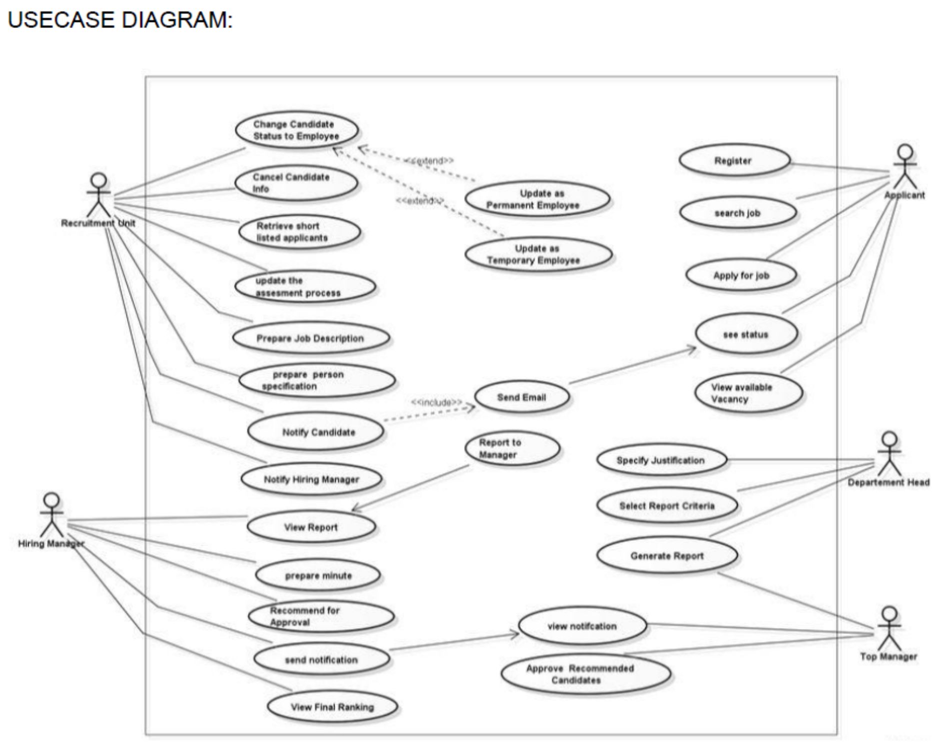


Figure 1: Use Case Diagram for Employee Management System

This diagram outlines the primary actors—HR Admin and Employees—and their interactions with the system across modules like onboarding, payroll, leave requests, and document verification.

2.2 Activity Diagram

EMPLOYEE MANAGEMENT

ACTIVITY DIAGRAM:

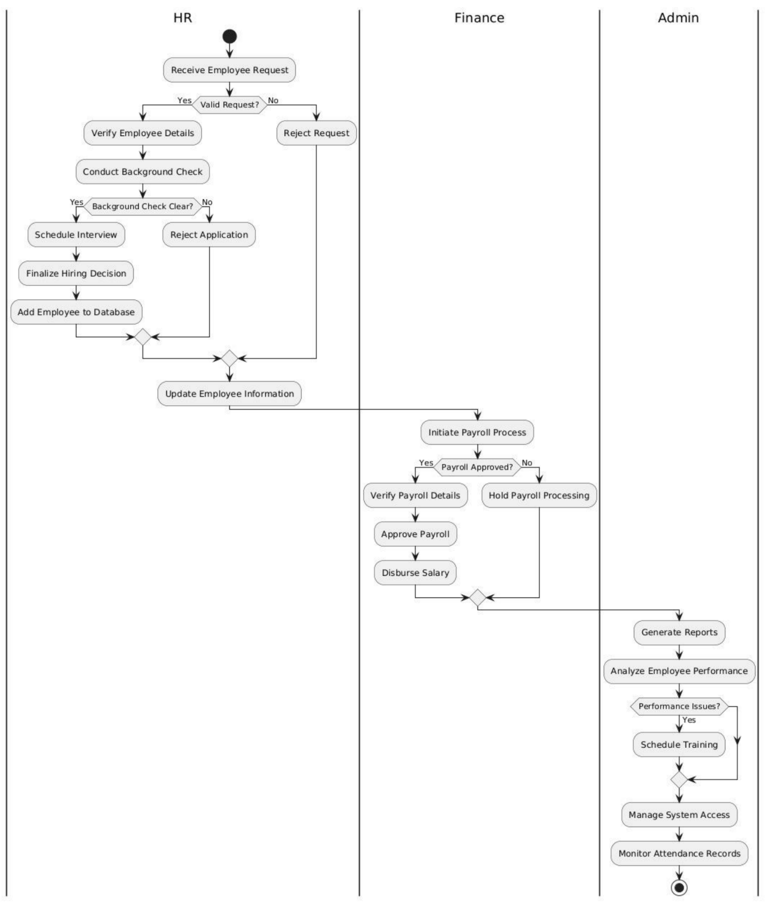


Figure 2: Activity Diagram

The activity flow highlights how an employee registers, uploads documents, and gets verified through a sequence of steps handled by both frontend and backend services.

2.3 State Diagram

STATE DIAGRAM:

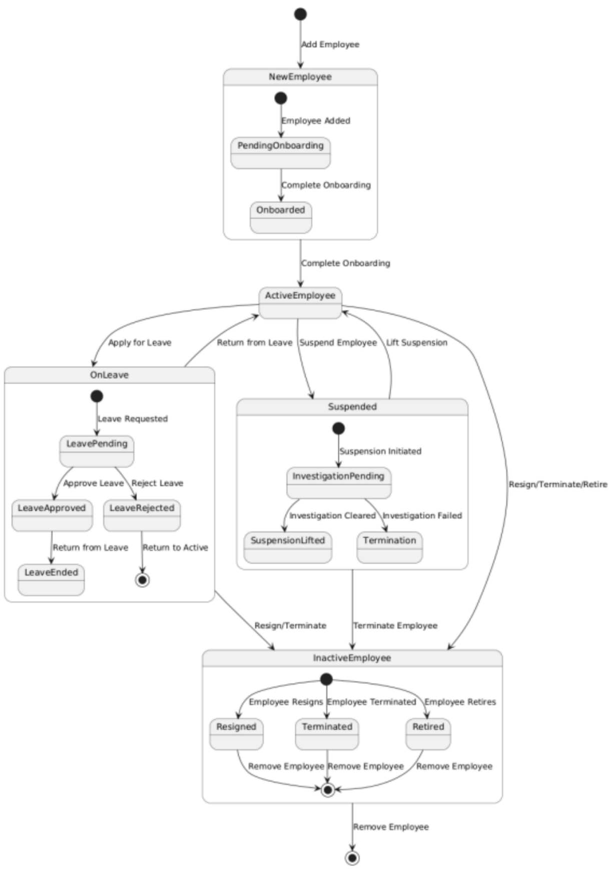


Figure 3: State Diagram

This diagram demonstrates various states an employee request can go through—such as submitted, under verification, approved, or rejected—during the background verification process.

2.4 Class Diagram

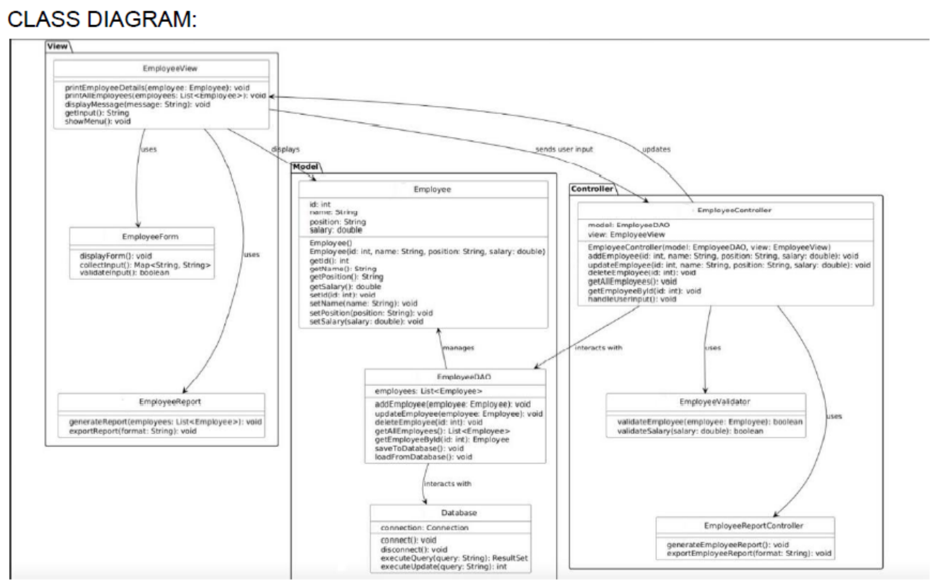

CLASS DIAGRAM:
View
EmployeeView
printEmployeeDetails(employee: Employee): void
printAllEmployees(employees: List<Employee>): void
displayMessage(message: String): void
getInput(): String
showMenu(): void
uses
displays
sends user input
updates
uses
EmployeeForm
displayForm(): void
collectInput(): Map<String, String>
validateInput(): boolean
EmployeeReport
generateReport(employees: List<Employee>): void
exportReport(format: String): void
Model
Employee
id: int
name: String
position: String
salary: double
Employee()
Employee(id: int, name: String, position: String, salary: double)
getId(): int
getName(): String
getPosition(): String
getSalary(): double
setId(id: int): void
setName(name: String): void
setPosition(position: String): void
setSalary(salary: double): void
manages
EmployeeDAO
employees: List<Employee>
addEmployee(employee: Employee): void
updateEmployee(employee: Employee): void
deleteEmployee(id: int): void
getAllEmployees(): List<Employee>
getEmployeeById(id: int): Employee
saveToDatabase(): void
loadFromDatabase(): void
interacts with
Database
connection: Connection
connect(): void
disconnect(): void
executeQuery(query: String): ResultSet
executeUpdate(query: String): int
Controller
EmployeeController
model: EmployeeDAO
view: EmployeeView
EmployeeController(model: EmployeeDAO, view: EmployeeView)
addEmployee(id: int, name: String, position: String, salary: double): void
updateEmployee(id: int, name: String, position: String, salary: double): void
deleteEmployee(id: int): void
getAllEmployees(): void
getEmployeeById(id: int): void
handleUserInput(): void
interacts with
uses
uses
EmployeeValidator
validateEmployee(employee: Employee): boolean
validateSalary(salary: double): boolean
EmployeeReportController
generateEmployeeReport(): void
exportEmployeeReport(format: String): void

Figure 4: Class Diagram

The class diagram maps core entities like `Employee`, `Candidate`, `Payroll`, and `Leave`, showcasing their attributes and relationships with controllers, services, and repositories.

3 MVC Architecture

The project implements the **Model-View-Controller (MVC)** architectural pattern for modularity and separation of concerns:

- **Model:** Encapsulates data and business entities such as `Employee`, `Candidate`, and `Leave`. These are Java POJOs annotated with `@Entity` and managed via JPA.
- **View:** Built using React.js, the view layer manages user input and displays data dynamically through reusable components.
- **Controller:** Spring Boot REST controllers like `EmployeeController` expose endpoints to receive user requests and delegate processing to the service layer.
- **Service:** The business logic is handled in services like `PayrollService` or `VerificationReques` which coordinate data flow and error handling.
- **Repository:** Interfaces like `EmployeeRepository` extend `JpaRepository`, providing database interaction methods and query abstraction.

Data Flow Example:

> React UI → REST API → EmployeeController → EmployeeService → EmployeeRepository → MySQL

This layered architecture promotes better testability, independent development, and maintainability.

—-

4 Design Patterns Used

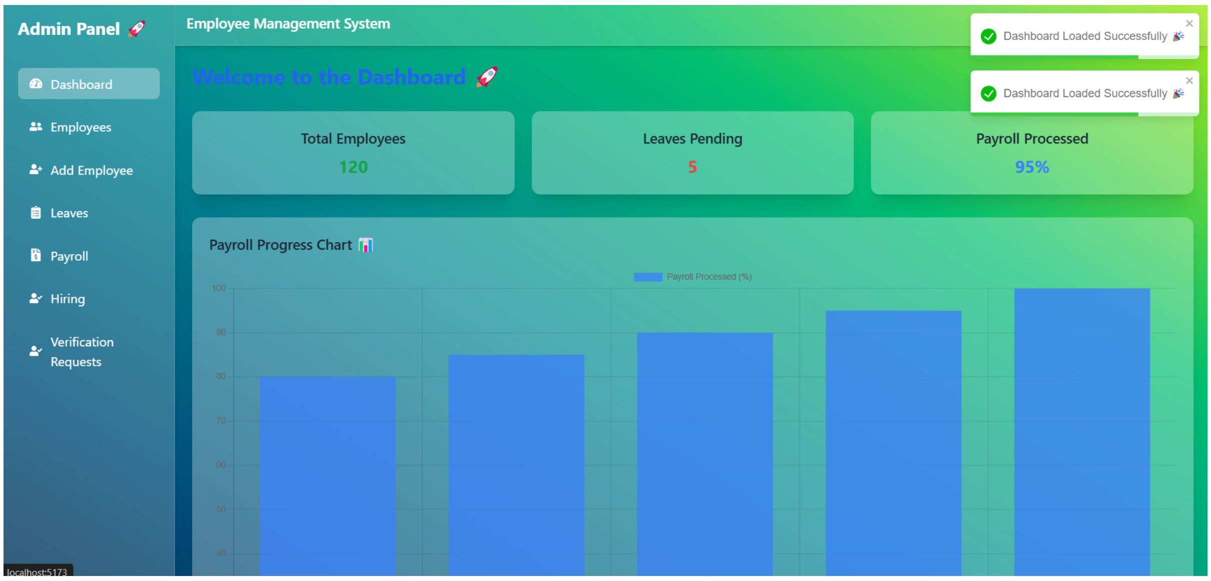
Several well-established object-oriented design patterns are applied within this system:

1. **Singleton Pattern:**
   - All service and controller beans are managed as singletons by the Spring container. For example, `EmployeeService` is auto-wired into controllers ensuring one consistent instance throughout the app.
2. **Factory Pattern:**
   - Spring's internal `ApplicationContext` functions as a factory, creating beans for annotated classes like `@Component`, `@Service`, and `@Repository`.
3. **Repository Pattern:**
   - Spring Data JPA promotes the repository pattern. Interfaces like `LeaveRepository` and `CandidateRepository` abstract SQL queries and enable clean database operations.
4. **Observer Pattern:**
   - This pattern is utilized to track system events. For example, upon document upload or background verification completion, events are published using Spring's `ApplicationEventPublisher`, triggering logging or email notifications.
5. **Builder Pattern:**
   - The frontend applies builder logic in assembling form data. When a user fills multiple inputs for creating an employee or placing an order, these values are step-by-step compiled into structured JSON requests using a builder-style logic.

These patterns work collectively to ensure loose coupling, high cohesion, and extensibility of the application.

5 Frontend Interface Screenshots

Dashboard



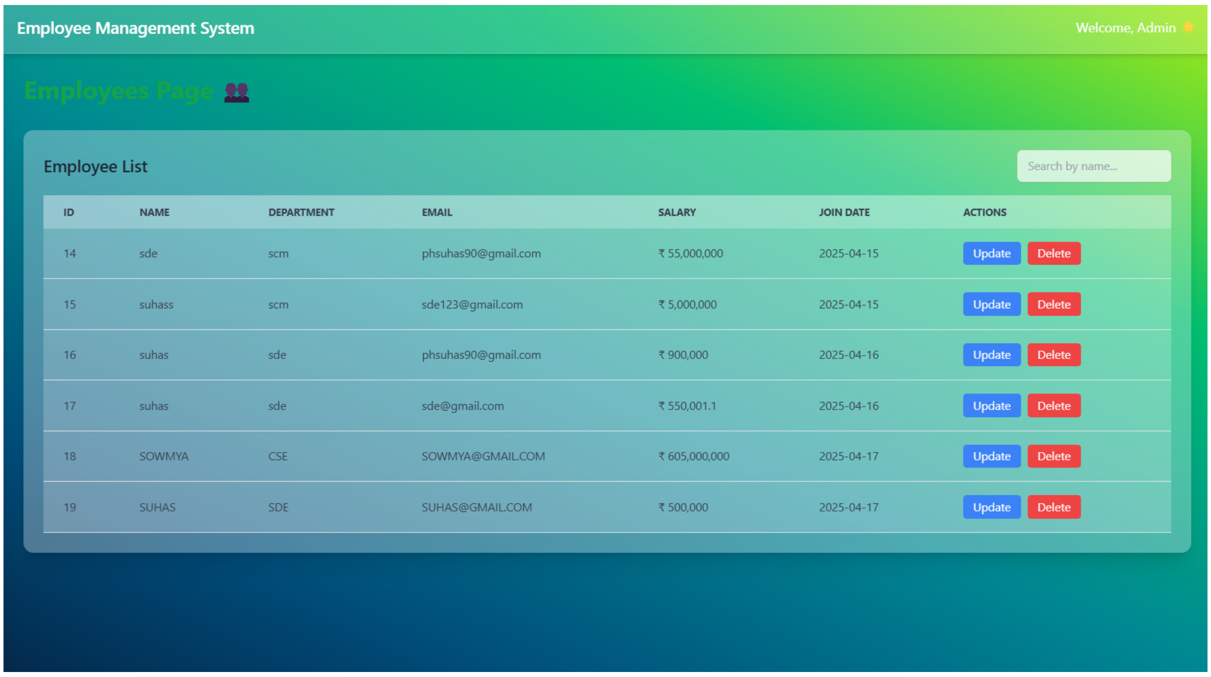
Dashboard Loaded Successfully 🎉

Dashboard Loaded Successfully 🎉

Total Employees	Leaves Pending	Payroll Processed
120	5	95%

Figure 5: Overview dashboard with employee stats and payroll chart

Employee List



Employee List

ID	NAME	DEPARTMENT	EMAIL	SALARY	JOIN DATE	ACTIONS
14	sde	scm	phsuhas90@gmail.com	₹ 55,000,000	2025-04-15	Update Delete
15	suhass	scm	sde123@gmail.com	₹ 5,000,000	2025-04-15	Update Delete
16	suhas	sde	phsuhas90@gmail.com	₹ 900,000	2025-04-16	Update Delete
17	suhas	sde	sde@gmail.com	₹ 550,001.1	2025-04-16	Update Delete
18	SOWMYA	CSE	SOWMYA@GMAIL.COM	₹ 605,000,000	2025-04-17	Update Delete
19	SUHAS	SDE	SUHAS@GMAIL.COM	₹ 500,000	2025-04-17	Update Delete

Figure 6: Manage employees - view, update, and delete actions

Add Employee

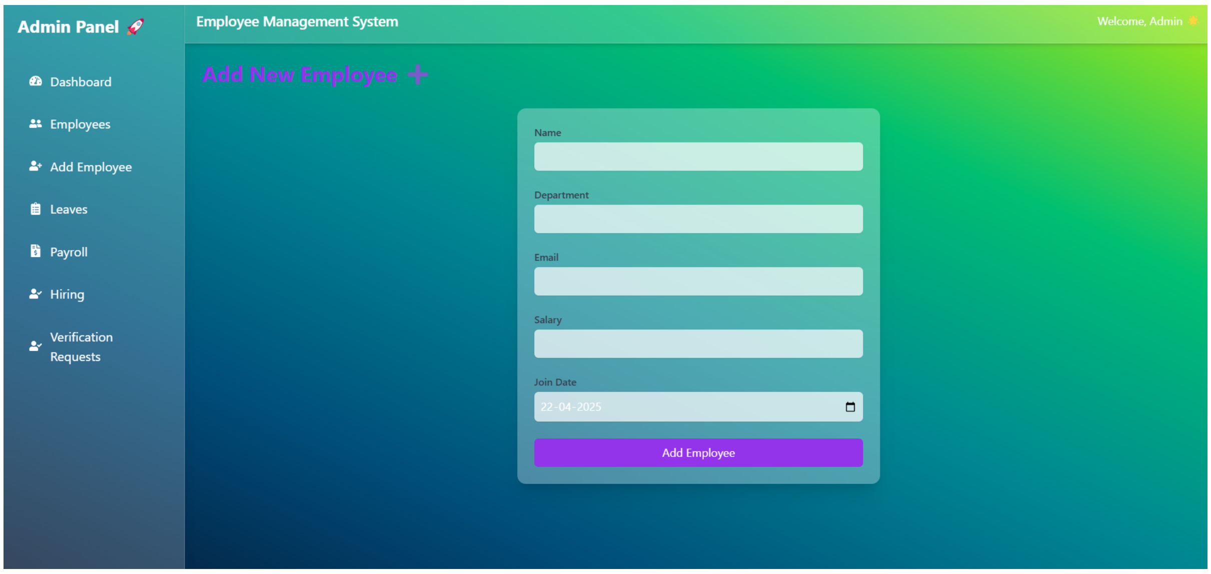
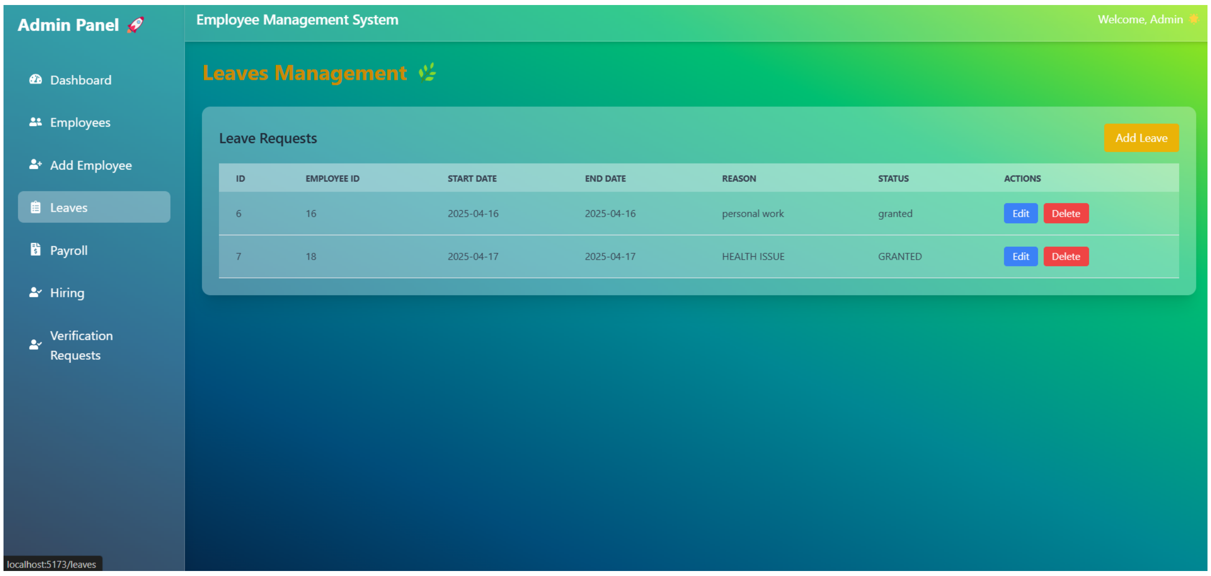


Figure 7: Admin form for adding new employees

Leave Management

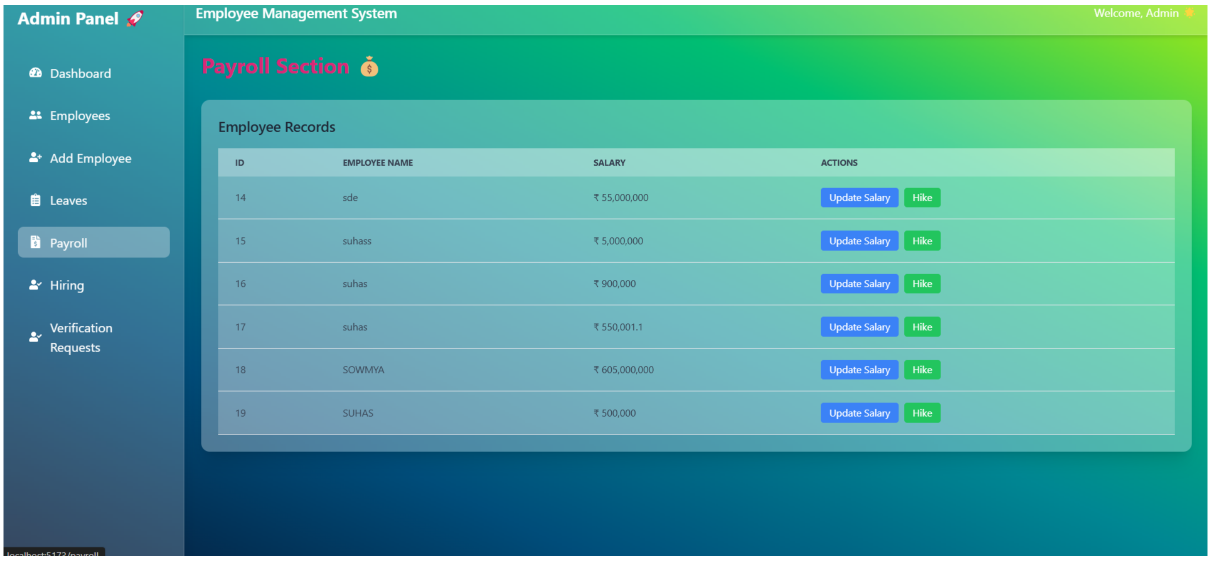


Leave Requests

ID	EMPLOYEE ID	START DATE	END DATE	REASON	STATUS	ACTIONS
6	16	2025-04-16	2025-04-16	personal work	granted	Edit Delete
7	18	2025-04-17	2025-04-17	HEALTH ISSUE	GRANTED	Edit Delete

Figure 8: Leave request handling interface

Payroll Section



Employee Records

ID	EMPLOYEE NAME	SALARY	ACTIONS
14	sde	₹ 55,000,000	Update Salary Hike
15	suhass	₹ 5,000,000	Update Salary Hike
16	suhas	₹ 900,000	Update Salary Hike
17	suhas	₹ 550,001.1	Update Salary Hike
18	SOWMYA	₹ 605,000,000	Update Salary Hike
19	SUHAS	₹ 500,000	Update Salary Hike

Figure 9: Salary update and hike management

Hiring Form

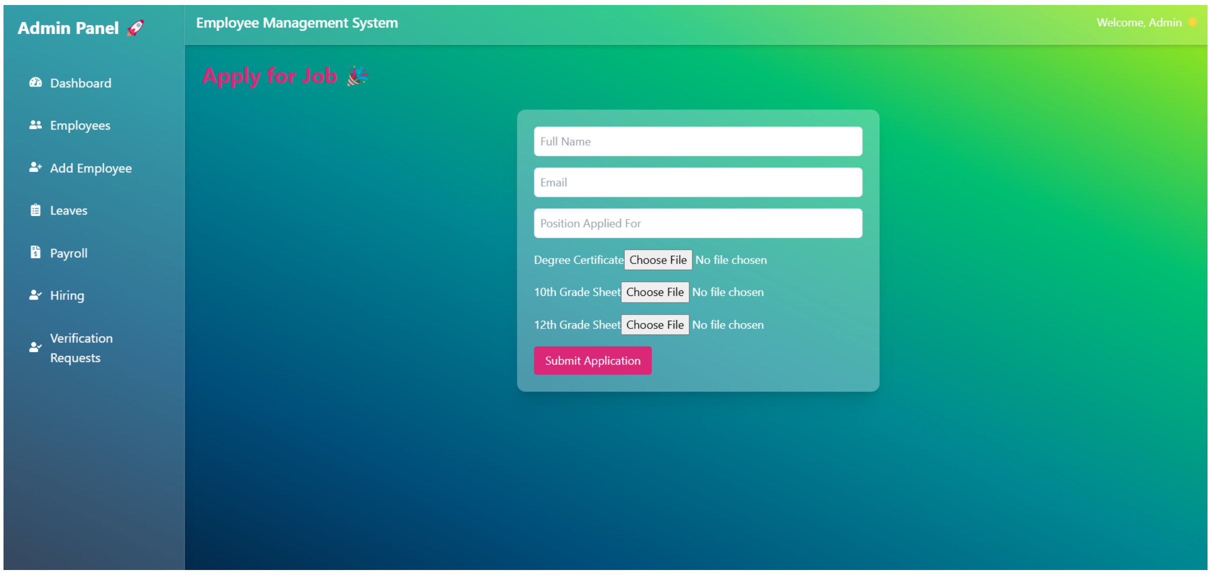
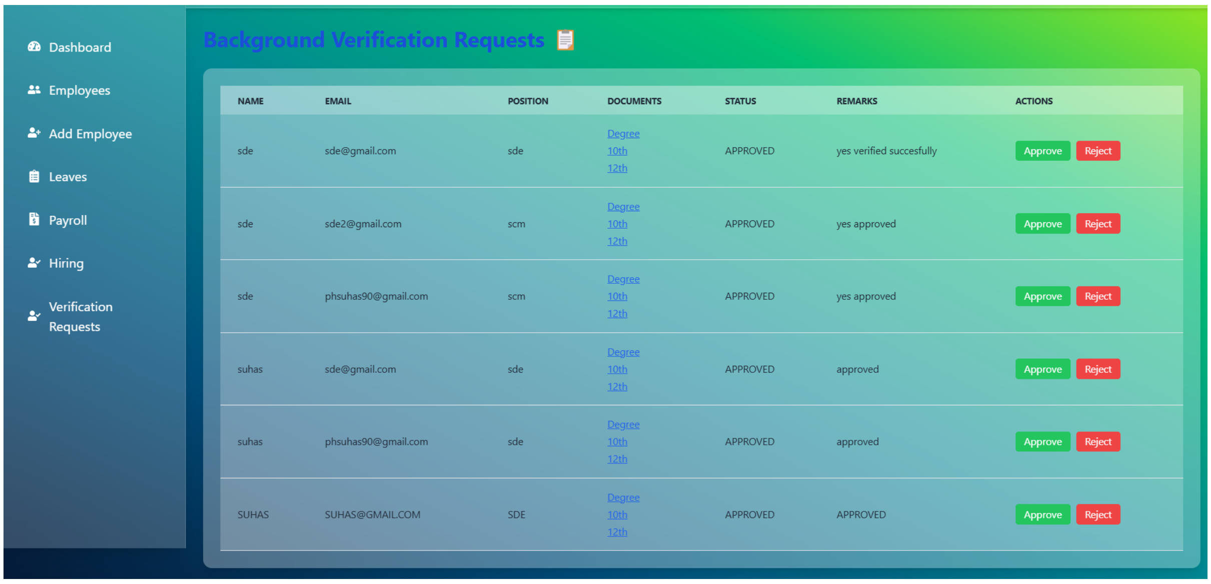


Figure 10: Job application with document uploads

Verification Requests



Background Verification Requests

NAME	EMAIL	POSITION	DOCUMENTS	STATUS	REMARKS	ACTIONS
sde	sde@gmail.com	sde	Degree 10th 12th	APPROVED	yes verified succesfully	Approve Reject
sde	sde2@gmail.com	scm	Degree 10th 12th	APPROVED	yes approved	Approve Reject
sde	phsuhas90@gmail.com	scm	Degree 10th 12th	APPROVED	yes approved	Approve Reject
suhas	sde@gmail.com	sde	Degree 10th 12th	APPROVED	approved	Approve Reject
suhas	phsuhas90@gmail.com	sde	Degree 10th 12th	APPROVED	approved	Approve Reject
SUHAS	SUHAS@GMAIL.COM	SDE	Degree 10th 12th	APPROVED	APPROVED	Approve Reject

Figure 11: Admin verification and document approval panel

6 Project Requirements

The following are the software and system requirements for developing and running the Employee Management System:

Software Requirements

- **Frontend:** React.js (v18+), Node.js (v18+), npm
- **Backend:** Java 17, Spring Boot (v3+), Maven
- **Database:** MySQL (v8.0+)
- **Version Control:** Git, GitHub
- **IDE:** Visual Studio Code (for both frontend and backend development)
- **Browser:** Chrome / Firefox / Edge

Hardware Requirements

- Minimum 8 GB RAM
- At least 1 GHz Processor
- 500 MB disk space for backend and database
- Node.js environment and Java JDK configured

Project Repository

The complete source code for the project is available at the following GitHub repository:
https://github.com/suhas-ph/OOAD$_{M}INI_{P}ROJECT.git$

7 Conclusion

This mini-project has provided us with a comprehensive understanding of full-stack web application development. We explored and applied the **Model-View-Controller (MVC)** architecture, several **design patterns**, and integrated technologies like React.js, Spring Boot, and MySQL.

We developed a functional and visually engaging employee management platform with real-time capabilities and modular structure. Through this experience, we gained valuable insight into building scalable software and following best practices in software design and engineering.

This hands-on project significantly improved our grasp of **Object-Oriented Analysis and Design**, making us confident in applying these principles to real-world systems.